IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 August 2022, accepted 26 August 2022, date of publication 31 August 2022, date of current version 9 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3203179

RESEARCH ARTICLE

High-Throughput Low Power Area Efficient 17-bit 2's Complement Multilayer Perceptron Components and Architecture for on-Chip Machine Learning in Implantable Devices

JAMES BRIAN ROMAINE AND MARIO PEREIRA MARTÍN
Department of Engineering, Universidad Loyola Andalucía, Dos Hermanas, 41704 Seville, Spain
Corresponding author: James Brian Romaine (jbromaine@uloyola.es)

This work was supported by the Universidad Loyola Andalucía.

ABSTRACT In this manuscript the authors, design new hardware efficient combinational building blocks for a Multi Layer Perceptron (MLP) unit which eliminates the need for hardware generic Digital Signal Processing (DSP) units and also eliminates the need for on-chip block RAMs (BRAMs). The components were designed to minimise power and area consumption without sacrificing throughput. All designs were validated in a Field Programmable Gate Array (FPGA) and compared against unrestricted *CPU-MATLAB* implementations. Furthermore, a (2,2,2,2) MLP with back propagation was implemented and tested in a FPGA showing a total hardware utilisation of just 3782 LUTs, and no DSP or BRAMs. The MLP was also built in a Application Specific Integrated Circuit (ASIC) using a 130 nm technology by *Skywater 130A*. The results show that the area occupation was just 0.12 mm^2 and consumed just 100 mW at 100 MHz input stimulus.

INDEX TERMS Low power, area optimization, integrated circuits, FPGA, neural networks, perceptron, deep learning.

I. INTRODUCTION

Machine learning is a branch of Artificial intelligence which has been around since the 1980s and is a technique which intends to make software applications gain accuracy in their predictions without specifically encoding the information and hence have a learning aspect. Machine Learning (ML) has come to play a crucial and almost transparent role in everyday life, whether it be image classification, real-time object detection in autonomous vehicles or pattern recognition for the detection of diseases. ML is being applied all over the world and has had great success, now as ML techniques become more and more complex so do their techniques such as Neural Networks (NN). This increasing complexity is leading to bloated generic hardware which make use of parallelism such as Graphical Processing Units (GPUs). Nonetheless, this hardware is power hungry and big, meaning it is not suitable for IoT based device applications or implantable bio-medical devices [1]. Typical ML techniques include supervised learning, in which the system is given the correct answer and is penalised if it chooses the wrong answer and unsupervised learning, which uses only input data and looks for specific patterns such as clusters. A typical ML structure is the Neural Network (NN). The NN is a loose representation of the neurons which fire in the human brain based on the sum of the inputs to the neuron which activates or not. These neurons are typically stacked into various layers providing fine tuning to the network which in turn increases accuracy in their predictions and classifications. NNs have been around since the 50s, however it wasn't really until the 1980s when the MOSFET transistors were introduced that the idea of NNs really took off. [2]

Now many years on, ML techniques have shown great promise in the detection and classification of diseases such

The associate editor coordinating the review of this manuscript and approving it for publication was Yiming Tang.

as Epilepsy (EP). EP is a chronic disorder of the brain which results in uncontrollable seizures ranging from slight jerking or muscle twitches to complete loss of motor-control and violent jerking lasting several minutes. The probability of premature death is actually three times higher in EP patients and affects an estimated 50 million people worldwide. It has been identified that EP occurs due to hyperactive neural circuits which begin to fire in synchrony, this can be seen on an Electroencephalography (EEG) recording as high amplitude local field potentials (LFPs) [3], [4]. Examples of ML in EP can be seen in [5], where the authors use ML techniques for diagnosing and prognosis of patients via Convolutions Neural Networks (CNNs) using neuroimages or in [6], where advanced ML techniques are used for the detection of Epilepsy via Electroencephalogram (EEG) signals captured via a scalp cap. Further examples can be found in [7] and [8]. However, due to the large size of the NNs and the non-optimised hardware these types of NNs would produce very high area and power consuming devices, which would not be useful for such applications as *in-vivo* implantable devices.

Implantable device for medicine have been around for many years in terms of pacemakers, insulin pumps, hip joints and more, but its not until recently that the idea of implementable microchips in the form of Application Specific Integrated Circuits (ASICs) has become a growing and very realistic theme. Firstly, this is partly due to the advances in semi-conductor technologies where the miniaturisation of complementary metal oxide semiconductors now range down to 7-5 nm in size approximately. This allows for more transistors per area hence increasing the processing power per area. Area occupation is an important factor in implantable devices, for example in cortical implants where the *ASIC*, sits below the cranium close to the delicate brain tissue there is limited room for big processors [9]. Secondly, the operating voltages are continuing to decline providing more processing power at lower power per area [10]. This aspect of implantable devices is possibly the most crucial since the continuous processing of *ASIC* devices produces excess heat which can damage cortical tissue if not properly managed. In [11], the authors noted that Computer Brain Interfaces which surpass 39° C can produce irreversible damage to the surrounding tissue area. To this end it is important for the *ASIC* designer to ensure the lowest power consumption possible. Both of these aspects can be reduce by cleverly designing the hardware to reduces the total number of transistors switching per operation or reduce the operating frequency. This can be achieved by avoiding generic hardware and controlling the maximum bus widths of the system.

Examples of cortical based *ASIC* implementations can be seen in [12], where the authors introduce an optimised Very Large Scale Integration of a brain state classifier using a 130 nm technology. The design could correctly classify epileptic seizures with up to 97% accuracy and consumed only 169 μJ per classification sitting on a total occupied area of just 2.55 mm by 1.3 mm. Another example of a Epileptic detection system can be seen in [13], where the authors try to create a trade-off between accuracy and power consumption which in this case was implemented in a 180 μm technology, consumed just 15 nW at 0.5V supply and sat on an area of just 0.05 mm^2. Furthermore, the throughput and overall latency of the system is also an important aspect when considering the detection and possible stimulation of diseases such as epilepsy where the seizures may spontaneously erupt. In which case, the system should be capable of classifying this information within micro seconds. This of course becomes a trade of between accuracy/latency/power and area consumption. Where high parallelism will provide higher throughput but will consume more area and power [14].

The miniaturisation, of technologies is also giving rise to the possibility to place ML classifiers on-chip providing fast, re-configurable and sensitive possibilities for the detection of diseases. The main problem with ML and DL techniques mainly comes from the total amount of complex operations which are needed. Which includes both long multiplications and divisions. These are known as the most complex operations in digital systems and tend to increase latency and can vastly increase the area and power consumption of a system which contains many multipliers or divisions [15]. This is especially true in NN since many of the common building blocks are based on tun-able elements. Field Programmable Gate Arrays (FPGAs) and programmable logic (PL) devices for example incorporate DSP blocks which incorporate very specific reconfigurable hardware blocks multipliers, pattern detectors and accumulators, in the Virtex-7 FPGA for example a standard DSP element consists of a 25 $\times$ 18 two's-complement multiplier, 48-bit accumulator, pattern detector, pre-adder and a Single-instruction-multiple-data (SIMD) arithmetic unit. Furthermore, many designs leverage the need for large BRAM memory units, where the Vitex-7 has a reconfigurable 16-32 KB RAM unit. BRAMs suffer from read latencies and hence are inferior to the speed of Look Up Tables (LUTs) [16].

Therefore, it is clear that important aspects such as the area occupied by the designed hardware, power consumption, parallelism and latencies are not prioritised in most systems instead leveraging components from the standard libraries making on-chip NNs unfeasible in real world applications due to high resource allocation and non-optimisation. Due to these factors, there is a need to reduce hardware resource utilisation in order to allow on-chip NNs in implantable devices whilst maintaining high throughput at low power costs. These types of reductions can be key to unlocking on-the-fly NN computing for diseases such as EP.

The following contributions are made in this manuscript:

- Firstly, the authors intent to alleviate the heavy power and area consumption of on-chip ML by designing and implementing low-cost digital hardware building blocks capable of being inserted into various other ML designs.These digital building blocks also give way to the possibility of building *ASIC* applications by

modifying and reducing the cost of multiplications and divisions.
- The digital building blocks are explained and designed in a Minized 7Z007S and further simulated comparing the results to unrestricted *MATLAB* based *CPU* implementations and each component is compared against the state of the art providing evidence of lower area and power consumption.
- As evidence of correct functionality the construction of a (2,2,2,2) feed forward topology is implemented in a FPGA and the results shown for a simple application.
- Lastly, the individual components are implemented into an ASIC 130nm technology implementation for area and power analysis and compared against the state of the art to show the benefits of these small and power efficient components.

The paper is constructed in the following manner, Section II, gives an introduction into some of the basic principles of NNs and deep layers including common Activation functions (AFs), propagation, weight/bias regularisation and loss functions. Section III, introduces the state of the art in implantable ML devices, Section IV, introduces the digital building blocks. Section V, shows the results from the *FPGA* implementations. Section VI, introduces the *ASIC* design including power and area consumption results, finishing with the conclusions in Section VII.

II. NEURAL NETS OVERVIEW

A. THE NEURON (PERCEPTRON)

In the human body neurons number in the billions, interconnecting which are activated by sum of the input voltages leading to an action potential (activation) or not based on a specific voltage threshold. In NN the neuron is a loose representation of its biological counterpart Fig. 1 shows a representation of a neuron where $input_j$ are the individual inputs to the neuron, w_j are the trainable adjustment weight (discussed further in sec II-E), $b1$, is an initial bias offset and σ (explained in more detail in sec II-C) is an AF. The function of the neuron can be denoted by equation (1). As the neurons potential increases or decreases the sum is calculated of which is then passed through an AF which represents the modelling of the action potential. However, in this case the AF can be more complex allowing for more complex non-linear data modelling [17].

$$Output(x) = \sigma(\sum_{j=1}^{T_x} w_{i,j} Input_j + b) \tag{1}$$

B. MULTI LAYER PERCEPTRON

An example of a NN can be seen in Fig. 2. Consisting of interconnected layers of neurons $N_{i,j}$, where the neural columns are donated by j and the neural row denoted by i. This particular NN has an input layer where the input values are constrained to the set of real values between -1 and $+1$. Furthermore, it has one hidden layer denoted h_1 with an activation function and an output layer denoted O_1 closley followed by a softmax activation for categorisation. The weights and biases for each neuron are denoted $w_{weight,layer}$. This basic example is what is known as a fully connected layer or Multi Layer Perceptron and are mainly used for non-linear model fitting and require back propagation to update the weights and bias values at the input of each neuron.

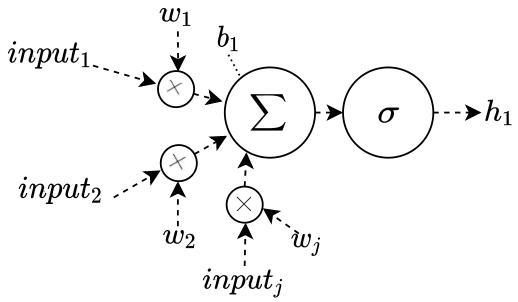


FIGURE 1. Example neuron, including inputs, weights, bias and activation.

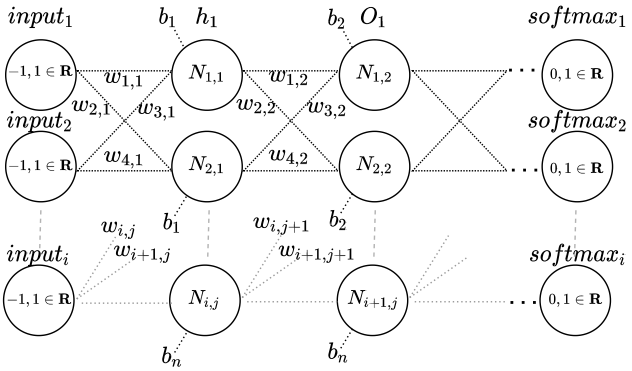


FIGURE 2. Example interconnected neurons, with one hidden layer with leaky-ReLu activation and an output layer with a soft-max activation.

C. ACTIVATION FUNCTIONS

Due to the non-linearity of most data and classification problems AFs are used which typically have a non-linear output such that from (1) we get, $\sigma(Output(x))$, where σ represents the AF. Below we introduce two of the main low area consuming AFs and a categorisation AF.

1) RECTIFIED LINEAR UNIT

The RelU AF is non-linear described by (2). This AF is by far the most commonly used due to its simplicity and is the most common AF in CNNs. Nonetheless, the RelU function is not as non-linear as the other functions making it less sensitive to non-linear input data. Furthermore, the output of the function is bounded as the maximum value of the neurons meaning large bus widths at the hardware-level and the absence of outputs for $x <= 0$, can give rise to dead neurons accumulating in the system. However the implementation remains easy and the derivative as seen in (3) and is equal to a step function

about zero [18].

$$Relu(x) = max(0, x) \tag{2}$$

$$f'(Relu(x)) = \begin{cases} 0, & \text{if } x <= 0 \\ 1, & \text{otherwise.} \end{cases} \tag{3}$$

2) RECTIFIED LINEAR UNIT LEAKY

The Leaky RelU AF is non-linear described by (4). This AF is very similar to the ReLu function however a small decay value C is placed on the values less than zero producing a leaky effect. This solves the problem of dead neurons. (5) is the derivative which is not differentiable at 0 [18].

$$Relu_l(x) = \begin{cases} x, & \text{if } x \geq 0 \\ C \cdot x, & \text{otherwise.} \end{cases} \tag{4}$$

$$f'(Relu_l(x)) = \begin{cases} 1, & \text{if } x > 0 \\ C, & x < 0 \end{cases} \tag{5}$$

3) SOFTMAX

The softmax AF is a normalised exponential function widely used as an output layer AF. Due to its probabilistic characteristics allows the outputs to be mapped to a probability distribution. Softmax is incorporated whenever more than one category at the output exists. A simple example would be the categorisation of animals such as cats, dogs, horses etc.. The equation for a softmax function can be seen in (6) [18].

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad for\ i = 1, 2, \ldots, K \tag{6}$$

These activation functions are just three of many AFs where other more non-linear functions such as the logistic function (LF) defined as: $\sigma(x) = \frac{1}{1+e^{-x}}$ or the *Tanh* AF: $tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = \frac{1-e^{-2x}}{1+e^{-2x}}$ exist however are much less hardware friendly and are used when high non-linearity is needed.

D. LOSS FUNCTIONS

Loss functions (LF) are used to estimate the error in a supervised learning system and quantifies how close the predicted result from the NN was with respect to what the actual value is. Some examples of loss functions include:

1) SQUARED ERROR

The squared error (SE) is a typical loss function as seen in (7) the quality estimator is always strictly positive decreasing as the error approaches zero. The derivative can be calculated as in (8). Where $\hat{y}_i$ is the expected value and y_i is the predicted value by the system and D is the total number of output neurons. [19]

$$SE = \frac{1}{2}\sum_{i=1}^{D}(\hat{y}_i - y_i)^2 \tag{7}$$

$$SE' = (\hat{y}_i - y_i) \tag{8}$$

2) CROSS-ENTROPY

In binary classification, where the number of classes M equals 2, Binary Cross-Entropy(BCE) can be calculated as per equation (9):

$$CE = -\ln(-(y\ln(\hat{y}) + (1-y)\ln(1-\hat{y}))) \tag{9}$$

If $M > 2$ (i.e. multi-class classification), we calculate a separate loss for each class label per observation and sum the result as per equation (10) [19].

$$CE = -\sum_{c=1}^{M} y_i \ln(\hat{y}_i) \tag{10}$$

and the derivative can be calculated as per (11)

$$CE' = \hat{y}_i - y_i \tag{11}$$

Again, Where $\hat{y}_i$ is the expected value and y_i is the predicted value by the system.

Other loss functions may include, Huber loss and regression [19].

Many other types of activation functions exist both linear and non. Some examples include, Linear function, GelU, Exponential linear unit [18].

E. BACK-PROPAGATION

Back propagation is the process in which the NN actually learns based on passed mistakes. In a supervised learning system this means providing the correct output to the softmax layer and using gradient decent to update the weights and biases based on the partial derivative error at each node in the network.

As an example of the PD of the network error for weights $w_{1,2}$ and $w_{1,1}$ in Fig. 2 would lead to the PD equations (as seen in Appendix VII) for each weight and bias assuming that the network consists of one hidden layer an input layer and output layer and a softmax classification [20].

F. GRADIENT DECENT

Gradient decent (GD) is a iterative optimiser used in NN to find the local minimum or maximum of a system. In (12) the GD can be used to minimise the cost of the function by updating the weights with the P.D of the error, where α is an adjustable parameter called the learning rate imposed on the system to speed up learning and/or to prevent over and under-fitting of the system [21].

$$w_{i,j}* = w_{i,j} - \alpha \cdot \frac{\partial E}{\partial w_{i,j}} \tag{12}$$

G. NORMALISATION

Normalisation is used in many cases due tot he exploding gradient affect in which the weights shoot of towards infinity. In order to counteract this the weights should be penalised each time they get too big or too small. The two most common methods are L1 and L2 normalisation and can be denoted by

equations (13) and (14) respectively.

$$Loss = Error(Y - \widehat{Y}) + \lambda \sum_{1}^{n} |w_i| \tag{13}$$

$$Loss = Error(Y - \widehat{Y}) + \lambda \sum_{1}^{n} w_i^2 \tag{14}$$

III. STATE OF THE ART

A. FPGA IMPLEMENTATIONS

Hardware implementations of NNs on *FPGA* has become the forefront for high speed custom application specific NNs. FPGAs consist of LUTs, D-type flip-flops and multiplexers which are programably selectable allowing the designer to correctly select bus-widths, create custom hardware and increase parallelism. Therfore, *FPGAs*, provide more advanced processing per operational power than even the most common NN based hardware platforms such as CPUs and Graphical Processing Units (GPUs). In fact, *FPGAs* can provide great increases in throughput, in [22], the authors indicated a speed up of approximately 144× of a MLP when compared to that of a typical modern day CPU with multiple cores and threads. Nonetheless, many *FPGA* implementations do not focus on minimisation techniques and indeed use many of the standard library implementations offered by the software. This in turn leads to a more generic system which consumes more power and area than necessary. In [23], the authors build a MLP for various bit lengths, and produces a 95% classification accuracy at 16-bits resolution and 6 perceptrons in the hidden layer and a (7,6,5) topology. The design was implemented in an *Artix 7 FPGA*, and utilised only 3466 LUTs however the design also leveraged the FPGAs digital signal processing units (DPS) utilising a total of 81 DSP units and 1069 sliced registers, which, as explained in Section I, use low throughput generic multipliers and division units. The estimated power consumption of the MLP was 120 mW. nother example can be found in, [24], where the authors try to alleviate the hardware burden of the *FPGAs* DSP units on NNs by implementing a MLP unit which incorporates a modified multiplier unit based on intelligent shift additions. The system occupied 1179 LUTs and 1385 Sliced registers for 10 perceptrons achieving a 50% resource reduction when compared to other systems. This advancement eliminates the DSP module increasing scalability. In [25], the authors build a 18-bit, (15,20,20,1) topology using a RelU AF. The hardware architecture combines concepts from matrix computation fundamentals, mixed serial-parallel computer architecture, and specific hardware availability in current FPGA devices as ALUs and distributed RAM. The system occupies 1267 sliced registers, and 1198 LUTs, the design does not incorporate any BRAMs however does use 22 of the FPGAs standard Virtex-7 DSP building blocks. In [26], a similar architecture can be found where the authors implement an MLP on a Virtex-7 FPGA. The design used a 24-bit 2's complement data format and used almost 219 DSP units for a (12,7,3) topology adn 2 BRAM units, again drawing on the generic hardware based on FPGAs. Finally, in [27], the authors implement a (4,2,4) topology autoencoder which uses only 1047 sliced registers and 1033 LUTs, however the design does draw on 5 DSP units.

B. MULTIPLIERS

As stated multiplications are the bed rock of MLPs requiring multiple multiplications per perceptron. To this end work such as in [28], show how approximate multipliers can save hardware whilst maintain accuracy. Some notable designs can be found in [29], where the authors use bit truncation. By truncating the LSBs of the data word a smaller multiplier can be used however at the expense of a slight decrease in performance. In [30], where a LSB search checks the bits for a 1 in the case that a 1 is detected all reaming LSB bits are set to one and the multiplication is made on the MSB. Further, multiplier implementations can be seen in [31], where an analogue multiplier is implemented in the form of a digital to analogue converter (DAC). In [32], a N X N analogue mixed-signal vector multiplier is built for neural computing where the inputs are digital pulses and the weights are controlled by current sources.

C. ACTIVATION FUNCTIONS

Another bottle neck of MLPs is are the non-linear activation functions. [33], gives an overview and some hardware reduction techniques for the implementations of these functions, including techniques such as *COordinate Rotation DIgital Computer* (CORDIC) implementations. CORDIC is a hardware friendly method for the implementation of functions and its roots date back as far as 1956. The CORDIC can calculate a wide range of functions, including trigonometric functions, by taking a vector v_i and rotating it in small positive or negative increments in a circular, linear or hyperbolic coordinate system [34].The CORDIC suffers from several key downfalls, firstly CORDIC is an iterative process leading to latency issues, furthermore it requires large look-up-tables in order to store relative phase angles. Other methods include the calculation of functions using Taylor's expansion (TE) which consists of calculating an infinite sum of a terms derivatives for a given sample [35], however suffer from complex divisions and large exponential values creating a heavy dependence on multipliers. In [36], the authors build an estimate of the softmax function by estimating e^x via LUT lookups and use the *FPGA* fabric DSP blocks to calculate multiplications and divisions in the system. The DSP units means non optimised hardware is used occupying non-necessary area. Using a 16-bit system and a *Virtex 6 FPGA*, the design occupies 300 LUTs, 558 sliced registers and 5 DSP units as well as 8 BRAMS. Another approximation can be found in [37]. In [38], the authors try to estimate the signmoid, tangent and Radial Basis Function (RBF) using a simplicity Canonical Piece-wise Linear model (SCPLm). In [39], the authors implement a hardware friendly softmax function which uses base splitting to store the results of the

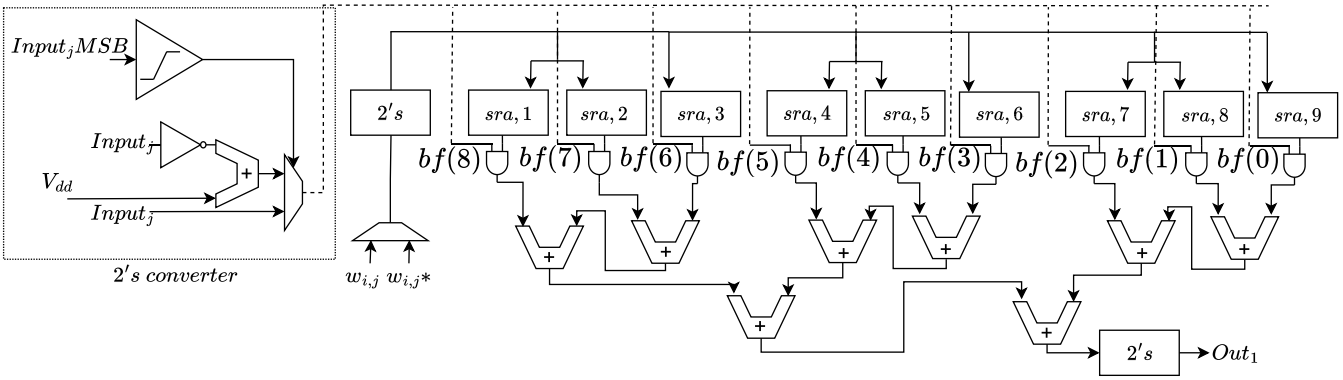


FIGURE 3. Block diagram of the SM unit with 2's complement converter and 9-bits of fractional precision.

exponential values in LUTs. In total their softmax implementation used 17870 LUTs and 16400 flip flops.

IV. PROPOSED ARCHITECTURE AND FPGA IMPLEMENTATION

The proposed architecture has been designed to minimise both area and power consumption, which are the two main drawbacks of NN implementations in implantable devices. This is achieved via the carefully design of hardware alleviating digital building blocks which do not compromise throughput. The architecture was designed as a 17-bit, signed, 2's complement system where the max fractional precision is 8-bits leading to a max decimal precision of 0.00390625. The integer is represented as 9-bits of data including a signed bit. The total bus width and hence precision is adaptable based on application specific tasks and should be adjusted to accommodate user specific applications. In this system, the input data must be normalised in the range $[-1,1]$ however the full data width was chosen as it is a common data width making comparisons more accessible.

A. SYSTEM MULTIPLIER UNIT

The system multiplier (SM) unit is one of the most important blocks in the system since the multiplier units are usually very expensive in terms of area and power cost. Moreover, NNs make extensive use of multipliers meaning reductions in this component can vastly alleviate the main drawbacks of NNs. The block designed in this manuscript is a low cost multiplier which maintains accuracy whilst increasing overall throughput. For reference the SM can be seen in Fig. 3. The SM unit takes advantage of 2^N right and left arithmetic shift operations which vastly increases operational throughput with very little cost in hardware. This 2^N based 2's complement shift only multiplier has the advantage of greatly reduced hardware due to the simplicity of the shifts as well as high throughput since all the shifts happen in parallel. This is contrary to most SM designs which intend to increase throughput via parallelism, however with the consequence of higher resource utilisation or vise a versa. The SM unit is constrained such that $-1 < Input_j < 1$ and works as follows: From Fig. 3 by using the individual fractional bits of $Input_j$ the sum of the shifts are calculated, where the amount of shift corresponds to the fractional bit of $Input_j$. A logical AND array is used to control the sum where, if the individual bit of the fractional part of $Input_j$ is active the logical AND array works as a pass-through key allowing the shifted right value to be summed to the final output. In the case that the $Input_j$ bit is deactivated the logical AND array provides an array of zeros to the sum not affecting the overall value.

As a simple example let $Input_j = 11000000_2$ and $w_{i,j} = 00000101.10100000_2$. This equates to $w_{i,j} \cdot Input_j = 5.625_{10} \cdot 0.625_{10} = 3.515625$. Since bits 7 and 5 are both active the final result at the output would be $((w_{i,j}, sra, 1) + 0 + (w_{i,j}, sra, 3) + 0 + 0 + 0 + 0 + 0) = (5.625_{10} \cdot 0.5_{10}) + (5.625_{10} \cdot 0.125_{10})$, where sra, x is a shift arithmetic right operation and x is the amount of shift to be applied. This has the advantage of maintaining high accuracy which, indeed, is only limited by the number of bits in the system. Furthermore, the system has a high throughput which can be as low as $t_{AND} + 3t_{sum}$, where t_{AND} is the combinational time delay of a single logical AND gate and t_{sum} is the combinational time delay of a full adder.

From Fig. 3, we should also note that the inputs to the SM unit must be converted from from 2's complement for values of $input_j < 0$ and converted back to 2's complement at the output. As an example for the input $input_j = -0.5$ and $w_{i,j} = -1.5$, the inputs would be converted to $input_j = +0.5$ and $w_{i,j} = -1.5$ hence the result would be a right arithmetic shift by 1 leading to -0.7 and a 2's conversion on the output would be necessary to bring the value back positive. To achieve this a simple 2's compliement converter system is placed on the input and output of the system and consists of a simple Most Significant Bit (MSB) comparator to detect the negative or positive input, a 1's complement converter and a full adder. To increase the accuracy of this SM unit we can simply add more fractional bits however this should be application specific and chosen to minimise area and power.

B. LEAKY RELU UNIT

The Leaky Relu unit is one of the least expensive blocks in terms of hardware cost. In this design the Leaky Relu block has been modified according to (15), in which two

TABLE 1. Over view of the FPGA hardware resource for each block.

Component	LUT-2	LUT-3	LUT-4	LUT-5	LUT-6	Carry-4
System multiplier	9	19	88	34	34	23
Leaky Relu unit	1	12	0	0	0	0
Exponential unit	19	20	74	37	36	24
Exponential unit2	60	60	150	121	150	66
SD unit1	255	61	82	58	48	89
SD unit2	70	93	89	158	225	85
Gradient decent	0	17	0	0	0	5
Gradient decent λ	23	12	15	0	22	2
Softmax	64	59	236	108	106	76
Perceptron	10	31	88	34	34	23
Feed forward	792	532	1240	594	658	312
Feed backwards	105	210	198	231	98	64

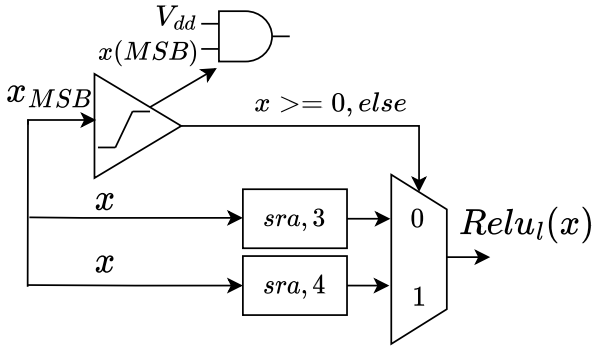


FIGURE 4. Block diagram Leaky Relu unit.

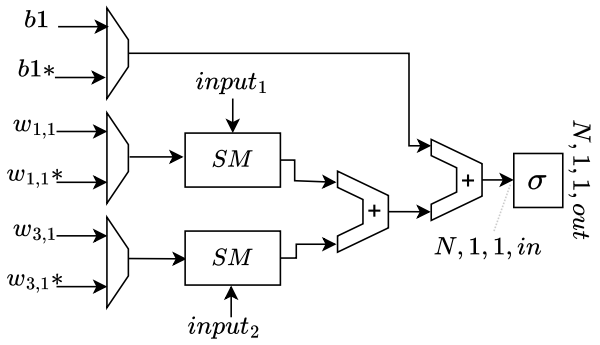


FIGURE 5. Block diagram perceptron.

decay variables $C1$ and $C2$ are introduced. $C1$ is a value which tries to constrain the output of the ReLu unit and should be user adjusted to such that it ensures the output does not increase above 1. In this way, the output is constrained similar to that of a sigmoid AF and allows the SM unit to be employed in deeper layers. Moreover, $C2$ is used to ensure small negative values and avoid dead neurons which is inline with the traditional Leaky ReLu such as in (4). This Relu unit should utilise the same amount of hardware resources as a standard leaky relu unit as long as the values of C1 and C2 are constrained to 2^N values. Nonetheless, the introduction of the C2 constraint further alleviates the system in future layers of the network. The hardware can be seen in Fig. 4, and incorporates a MSB comparator of 1-bit which checks the signed bit of the data word, if 0 it means that the value is above or equal to 0 otherwise the value is negative. This comparator can be implemented as a simple logical AND gate. A multiplexer on the output then pushes either $C1 \cdot x$ or $C2 \cdot x$ to the output. $C1$ and $C2$ decay rates should be selected as 2^N values such that they can be implemented as shift only values The latency of the system is calculated simply as the latency of the multiplexer t_{mux}, since the decay values are hard-wired connections.

$$Relu_l(x) = \begin{cases} C1 \cdot x, & \text{if } x \geq 0 \\ C2 \cdot x, & \text{otherwise.} \end{cases} \tag{15}$$

C. NEURON STRUCTURE

Now that we have introduced the AF and SM we introduce the hardware for a simple 2-input perceptron. Fig. 5, shows an overview of the hardware used to construct the neuron $N_{1,1}$ in the hidden layer of the NN shown in Fig. 2. The hardware corresponds to (1) where two SM units from Section IV-A are used to multiply the weights with the corresponding inputs. Moreover, two adder units are used to sum the results of the SM units and also sum the bias value before finally passing the output to our modified ReLu unit as seen in Section IV-B. Since the biases and weights will be updated during the training process each weight and bias must pass thorough a multiplexer where the symbol $*$ represents the updated weight or bias. In total the latency of a single perceptron is $t_{SM} + 2t_{sum} + t_{ReLu}$, where t_{ReLu} is the combinational delay of our leaky ReLu unit.

The neuron structure in this case is greatly simplified by the use of the SM unit which as we will see in future section the SM unit can be hardwired. This means that per perceptron we use no multipliers at all allowing for high throughput and low power consumption.

D. EXPONENTIAL UNIT

To estimate the exponential function we can use the taylors expansion (TE) where, for a function $f : \mathbf{R} \to \mathbf{R}$ k-times differentiable at the point $x = a$ $a \in \mathbf{R}$, Taylor's theorem states that there exists a function $h_k : \mathbf{R} \to \mathbf{R}$ such that:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{k)}(a)}{k!}(x-a)^k + h_k(x)(x-a)^k \tag{16}$$

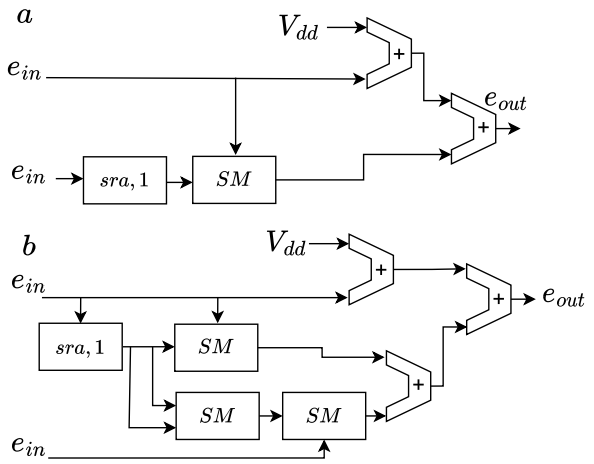


FIGURE 6. Block diagram exponential estimation unit (a): 3 - terms TE and (B): 4 - terms TE.

where $\lim_{x \to a} h_k(x) = 0$ and $f^{i)}$ is the ith derivative of the function $f(x)$.

Taking the first three terms and rearranging the equation we can get the estimation for e^x as per (17). From Fig 6 a, we can see the hardware configuration where two full adders are used to sum each term. The $\frac{x^2}{2}$ term is separated into two calculations firstly, $\frac{x}{2}$, which is handled via a shift right arithmetic similar to that in sec IV-A, secondly the multiplication of this term with x, note that since x comes from the adapted leaky ReLu block (as in sec IV-B) the maximum output value is scaled to less than 1 and hence the multiplication can be made using the adapted system multiplier from sec IV-A. This exponential unit uses very little hardware and is further adopted into blocks such as the softmax AF.

$$e^x = 1 + x + \frac{x^2}{2} = 1 + x + (\frac{x}{1} \cdot \frac{x}{2}). \tag{17}$$

The accuracy of the exponential unit can be improved by the addition of further terms, as an example in (18), the expansion for an extra term $\frac{x^3}{6}$ is shown, in the case of hardware this is implemented and shown in Fig. 6 b, where the new term is further broken down to allow for shifts of 2^N. However, $\frac{x^3}{6}$ cannot be fully broken down into perfect shifts such as $\frac{x}{3} \cdot \frac{x}{2} \cdot \frac{x}{1}$ and so is estimated to $\frac{x}{2} \cdot \frac{x}{2} \cdot \frac{x}{1}$. In the case of the hardware this incorporates an additional full adder and two SM units.

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} = 1 + x + (\frac{x}{1} \cdot \frac{x}{2}) + (\frac{x}{2} \cdot \frac{x}{2} \cdot x). \tag{18}$$

With respect to the latency the first exponential unit can update its output every $2t_{add} + t_{SM}$, where t_{SM}, is the combinational delay of the SM block in section IV-A. Since the second exponential incorporates more SM units the latency increases to $2t_{add} + 2t_{SM}$.

Both of these implementations force the TE series into a 2^N shift, which again like the previous blocks can be handled via simple and fast right shift arithmetic operations. Furthermore, this method allows for the SM unit block to be further implemented which as explained has high throughput and low area usage greatly reducing the size of the exponential units. If another method was used it is likely that the exponential units would consist of large generic based multipliers and DSP units.

E. SYSTEM DIVIDER UNIT

The SD unit is a crucial block in the softmax AF as described in Section IV-E. In this section the authors will introduce two possible implementations. The first is implemented as $\frac{1}{x} \cdot y$ and is based on a piece-wise-linear approximation (PWL). The PWL approximations were calculated as per $\frac{1}{x} = mx + c$ where the individual cases of m and c were calculated as per equation (19). Note that the maximum value of x in this case is $x < 10$ and the SD unit should be adapted as per application needs.

$$m, c = \begin{cases} -1010, 110.1 & \text{if } x >= 0.05 \text{ and } x < 0.1 \\ -43.4, 13.4 & \text{if } x >= 0.1 \text{ and } x < 0.2 \\ -15.4, 7.84 & \text{if } x >= 0.2 \text{ and } x < 0.3 \\ -7.9, 5.59 & \text{if } x >= 0.3 \text{ and } x < 0.4 \\ -2.38, 3.38 & \text{if } x >= 0.4 \text{ and } x < 1 \\ -0.68, 1.68 & \text{if } x >= 1 \text{ and } x < 1.5 \\ -0.32, 1.14 & \text{if } x >= 1.5 \text{ and } x < 2 \\ -0.17, 0.84 & \text{if } x >= 2 \text{ and } x < 3 \\ -0.08, 0.57 & \text{if } x >= 3 \text{ and } x < 4 \\ -0.05, 0.45 & \text{if } x >= 4 \text{ and } x < 5 \\ -0.034, 0.37 & \text{if } x >= 5 \text{ and } x < 6 \\ -0.023, 0.3 & \text{if } x >= 6 \text{ and } x < 7 \\ -0.018, 0.269 & \text{if } x >= 7 \text{ and } x < 8 \\ -0.015, 0.245 & \text{if } x >= 8 \text{ and } x < 9 \\ -0.01, 0.2 & \text{if } x >= 9 \text{ and } x < 10 \end{cases} \tag{19}$$

The hardware implementation of the PWL approximation can be seen in Fig. 7. Here we can note that several compartor units are needed to calculate both the fractional and integer parts of the input value to conform with the specific conditions set out by (19), where x_f is the fractional part and x_i is the integer part. A N to 4 decoder unit is then used to select the constant m, c values which are stored in LUTs, and routed via twin multiplexers. To finish we multiply m by the input via a SM unit and sum the result to c via a full adder. This system maintains very high throughput and can be calculated in as little as $t_{comp}+t_{dec}+t_{sum}+t_{SM}$, where t_{comp}, t_{dec}, t_{add}, t_{SM} are the combinational delays of a comparator, decoder, full adder and system multiplier respectively. It is important to note that in our application specific system the ReLu maximum output should be constrained to 1 and hence any m, c values above 1 can be eliminated from the hardware greatly reducing resources. In this case they have been left in to show the possible scale-ability of the PWL design, however in said case which the value of x increases above 1 the SM unit would

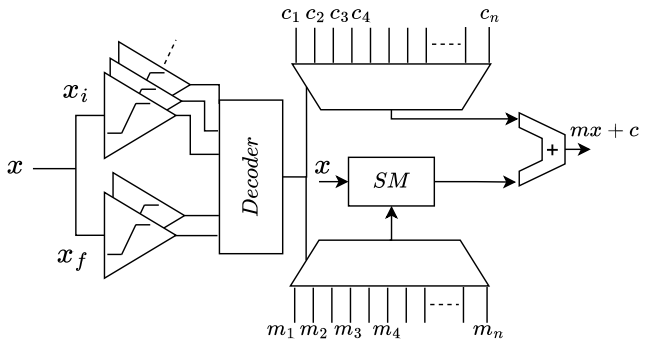


FIGURE 7. Block diagram SD unit.

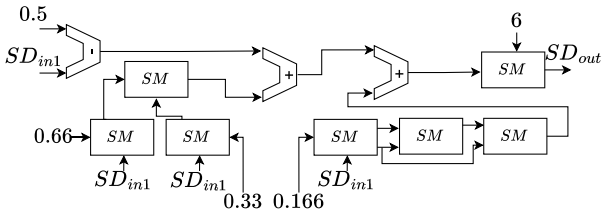


FIGURE 8. Block diagram SD unit alternative.

need to be replaced with a DSP core unit. This unit increases overall throughput by eliminating the repetitive subtractions which are normally needed in more generic systems.

For higher precision applications another possibility for implementing $\frac{1}{x}$ can be seen in Fig. 8 and is based on the equation (20),

$$1/x = \sum_{n=0}^{\infty}(-1)^n(-1+x)^n, \quad for\ abs(-1+x) < 1 \quad (20)$$

applying the first 4 terms equation (20) can be re-arranged to (21).

$$1/x = (-1)\cdot(1-x)^0 + (-1)\cdot(1-x)^1 + (-1)\cdot(1-x)^2 + (-1)\cdot(1-x)^3 \quad (21)$$

This can be further simplified to (22).

$$\begin{aligned} 1/x &= 3 - 6x + 4x^2 - x^3 \\ &\approx 6\cdot(0.5-x) + (0.66x\cdot 0.33x) \\ &\quad -(0.166x\cdot 0.166x\cdot 0.166x) \end{aligned} \quad (22)$$

To implement (22) in hardware extensive use of the SM is incorporated, requiring a total of 7, hence, increasing hardware utilisation.

F. SOFTMAX UNIT

The softmax unit can be seen in Fig. 9 which as per (6), makes heavy use of the exponential function, SM unit and SD unit. To this end the input to the softmax unit calculates the exponential values of sm_{in1} and sm_{in2} using twin exponential blocks as described in sec. IV-D. Since a division is needed here a SD unit and a SM unit are used such that we calculate:

$$\sigma(x_i) = e^{x_i}\cdot\frac{1}{\sum_{j=1}^{K} e^{x_j}} \quad (23)$$

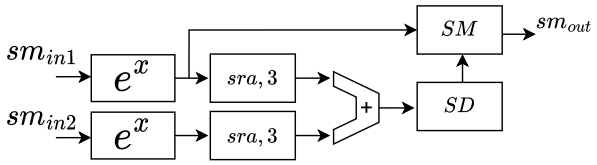


FIGURE 9. Block diagram Softmax unit.

Due to the restriction placed by (20), a shift right by 3 is applied to the previously calculated exponential values such that it provides a scaling factor $e^x < 1$ This increases the number of neurons that we can actively accommodate in each layer. Note that in statistical manner this scaling does not change the overall prediction, however it does limit the number of neurons in the output layer if the bus width is not increased.

In this case it is clear that the latency is $t_{exp} + t_{sum} + t_{SD} + t_{SM}$. This makes this block one of the most computationally hungry and large blocks, since a softmax block should be employed for each row of neurons user should decide if it is necessary.

The softmax block takes advantage of the previously designed exponential unit, SD unit and SM unit meaning that the hardware reductions and high throughput are carried over. This block reduces the need for CORDIC like repetitive algorithms and whilst marinating hardware reductions.

G. GRADIENT DECENT

The GD unit is simple and as per (12), the only complication is the multiplication by the learning rate α. The hyper-parameter can be set easily to a power of two meaning a simple shift right arithmetic block can be applied. An example of the GD block can be seen in Fig 10, where, in this case as per (24), we employ a L1 normalisation technique as seen in (13). In this case a MSB comparator is used to check if the weights are above or below zero and applies a addition or subtraction of λ based on the current weight via a multiplexer. The λ value tries to force weights above zero back towards zero and weights less than zero back towards zero. This has the benefit of allowing more room for the SM unit and the precision of the system to operate and furthermore ensures over-training does not occur.

$$w_{i,j}* = \begin{cases} (w-\lambda) - \dfrac{\partial E}{\partial w_{i,j}}, & \text{if } w > 0 \\ (w+\lambda) - \dfrac{\partial E}{\partial w_{i,j}}, & \text{if } w < 0 \end{cases} \quad (24)$$

H. BACK-PROPAGATION

The back propagation hardware is usually difficult, large and non generic since the back propagation depends on the network structure. In this manuscript the authors show an example of how our hardware can be used to implement a back pass. The example is based on updating the weights for

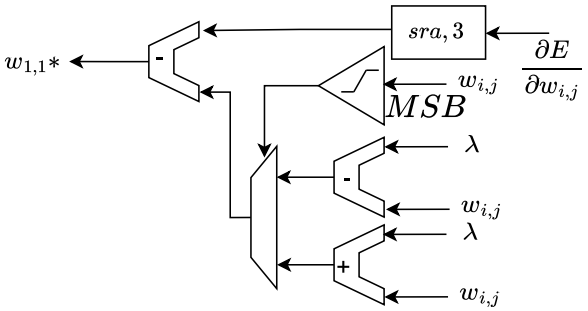


FIGURE 10. Block diagram GD unit.

$w_{1,2}$ and $w_{1,1}$ as seen in Fig. 2 and implemented as the PD from (28) and (32).

Fig. 11, shows an example of the hardware where (25), is implemented simply using a subtracter unit and a SM. to calculate (26) and (27) duplicate hardware is needed where 3 SM units calculate the multiplications before being summed together via a full adder to calculate the total error (32). Finally a GD unit from Section IV-G is used to update the weights accordingly based on the error. Using our SM unit we can note that as long as the $input_j$ value of the SM unit remains in the range $0 < input_j < 1$ we can employ the unit with no scaling.

$$\frac{\partial E}{\partial w_{1,2}} = (y_1 - \hat{y_1}) \cdot N_{1,2,out} \tag{25}$$

From (29),

$$\frac{\partial E1}{\partial w_{1,1}} = (y_1 - \hat{y_1}) \cdot w_{1,2} \cdot N_{1,1,out} \cdot input_1 \tag{26}$$

$$\frac{\partial E2}{\partial w_{1,1}} = (y_2 - \hat{y_2}) \cdot w_{2,2} \cdot N_{1,1,out} \cdot input_1 \tag{27}$$

An important note is that the latency of the back pass in this system is $4t_{SM} + t_{sum} + 2t_{sub}$, this will increase as the number of hidden layers increases but does not necessarily increase if the number of rows increases due to parallelism in the hardware.

I. HARDWARE REDUCTION TECHNIQUES

1) SHIFTS

The authors proposed architecture relies heavily on shift right and shift left arithmetic operations in order to provide us with a multiplication/division estimation. Reducing this block would greatly reduce hardware utilisation whilst maintaining high-throughput for the system. A shift right/left arithmetic operation consists of basic bit-shifting. This can be achieved synchronously using memory elements or asynchronously as a simple wired connection.

Fig. 12, shows an example of this where for a right shift by 1 we can hard-wire the most significant bit (msb) $b(16)$ from the input to bit $bo(15)$ of the output and $bo(16)$ of the output also receives $b(16)$ more generically put $b(N) \rightarrow bo(N-1)$, $bo(N) \rightarrow b(N)$. In the case of a right shift by 2, the input msb $b(N)$ connects to output $bo(N-2)$, $bo(N)$ and $bo(N-1)$ of the output both receive $b(N)$, note that the least significant bits of the input are lost, however this is the same in the case of logical shift registers. A shift left would be the same only prioritising the LSB units. Note that the latency of this is zero hence it is just a wire meaning we can ignore this block from the calculations of latency.

As a hardware reduction example this would reduce our SM module to 8 x adders and 136 AND gates.

V. RESULTS

The FPGA implementations were made using *VHDL* in the *VIVADO* environment and uploaded in an *Avent Zynq 7Z007S development board* using a default synthesising strategy. Note that the *FPGA* was chosen as a main strategy platform for testing as it is adaptable and allows for rapid application development. The *FPGA*, is the first stepping stone to full *ASIC* implementation.

A. SM UNIT

Fig. 13, shows an example of the FPGA results for the SM unit assuming a constant input of positive 1.5 and a multiplier input with a range of 0 to 0.99. A unrestricted CPU *MATLAB*, implementation has also been plotted for reference. Note that the SM FPGA implementation provides almost perfect output response, which is to be expected due to the hardware implementation and scaling techniques and produces a MSE of just 2.4661^{-05}. The SM hardware as per table 1, shows the full break down of LUTs used and shows that this SM unit utilises only 184 mixed-sized LUTs in total. Furthermore, when compared to other approximation multipliers as seen in table 2, our design uses less resources as well as provides less delay due to the single shift operation and logical AND gate key-pass.

B. RELU UNIT

Fig.14, shows an example output of the modified leaky Relu block taken from the *FPGA* implementation. In this scenario, $C1 = 0.125$ and $C2 = 0.0625$. For input values ranging from -8 to $+8$ the output range is hence allowed to swing from approximately -0.5 to 1 respectively. Note that the negative leaky section of the Relu unit approaches -1 at a slower rate due to the decay rate being lower. For our system $C1$ was chosen to allow the maximum input neuron sum to reach a maximum of $+8$ however this should be user adjusted. Table 3, shows a comparison between our design and an implementation in a Ultrascale 9 FPGA. Note that our design only uses 12 LUTs whilst the other design leverages 2 DSP units. Furthermore, due to the fixed shift values for C1 and C2 (to be user defined) our design has very little latency with the critical path just 0.9 ns.

Table 1, shows that the ReLu unit uses just 13 LUTs, 12 of which are 3-input LUTs.

C. EXPONENTIAL UNITS

Fig. 15, shows the output results from the *FPGA* implementation of the two exponential units described above in

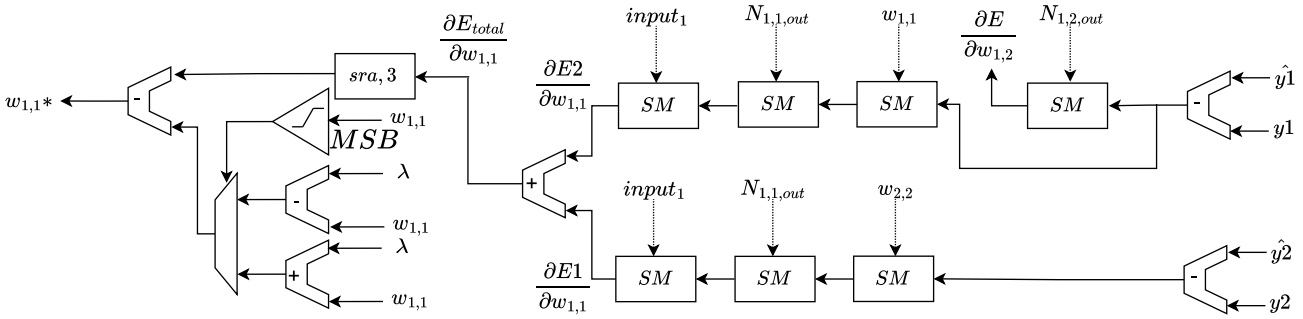


FIGURE 11. Example of the back pass hardware utilising our SD unit for $w_{1,1}, w_{1,2}$.

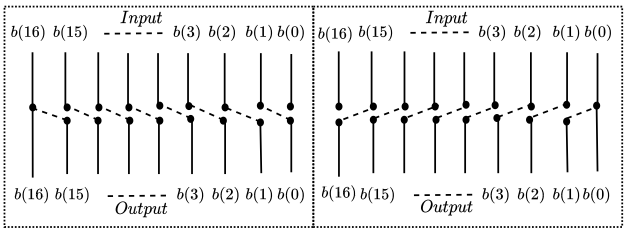


FIGURE 12. Shift right arithmetic (left) and shift left arithmetic (right) wiring to alleviate. hardware

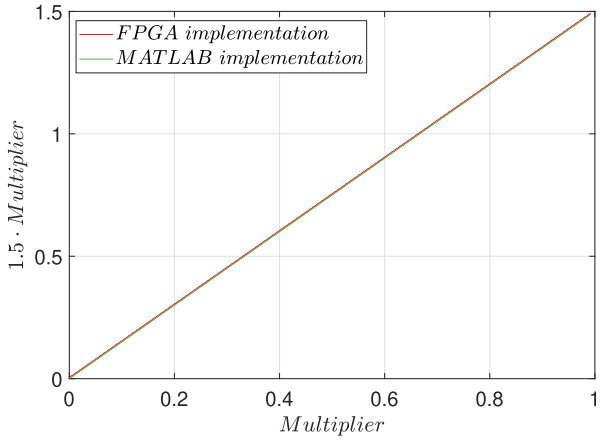


FIGURE 13. SM unit results for inputs ranging from 0 to 1 and a constant multipler of 1.5.

TABLE 2. Comparative table showing the FPGA resources of different multiplier implementations.

Ref	bits	Registers	LUTs	DPS	Delay(ns)	FPGA
This design	17	0	184	0	1.8	Zynq 7Z007S
[40]	16	0	208	0	5.28	Virtex-7
[41]	16	0	330	22	3.1	Spartan-6 FPGA

Section IV-D. The figure shows the output results for positive input values ranging from 0 to 0.99. When plotted against a unrestricted CPU *MATLAB* implementation it is clear to see that variations exist in both *FPGA* implementations. Moreover, it is clear that the second *FPGA* implementation produces a closer estimation resulting in a mean squared error (MSE) of 0.0041, whilst the the first FPGA implementation produced a MSE of 0.01.

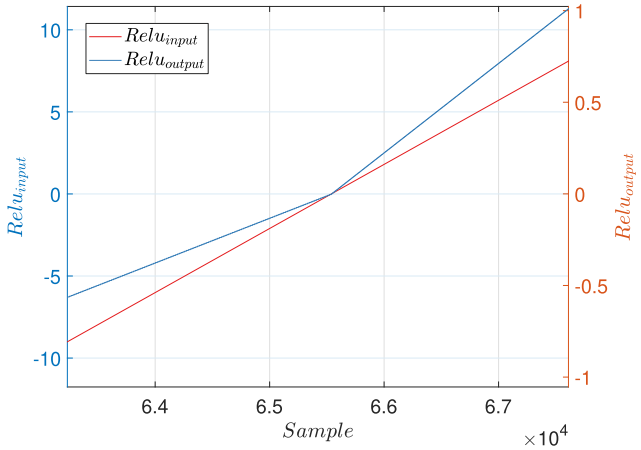


FIGURE 14. FPGA simulation of the Leaky Relu unit inputs ranging from −10 to 10.

TABLE 3. Comparative table showing the FPGA resources of different RelU implementations.

Ref	bits	Registers	LUTs	DPS	Delay(ns)	FPGA
This design	0	0	12	0	0.9	Zynq 7Z007S
[42]	0	0	2	2	12	Ultrascale 9

Fig. 16 shows the output results for negative input values ranging from -1 to 0. In this case the biggest errors occur the more negative the input value. Nonetheless, the second *FPGA* implementation again, clearly achieves higher accuracy, producing an MSE of 0.0012 whilst the first *FPGA* implementation produced an MSE of 0.0017. Table 1, shows the number of LUTs used by the *FPGA* in both cases where the fist and second exponential unit use just 186 and 541 mixed-size LUTs respectively. Similarly to the SM unit the precision is based on the number of TE terms incorporated into the design and can be expanded to gain higher precision based on application specific needs. In table 4, the resources of the two exponential units are compared to the state of the art. Once again due to the lower dependency on generic multiplications the TE exponential uses a lot less resources not leveraging any DSP units. Furthermore, the delays are smaller again due mainly to the fixed 2^N shifts and hardwired multiplications.

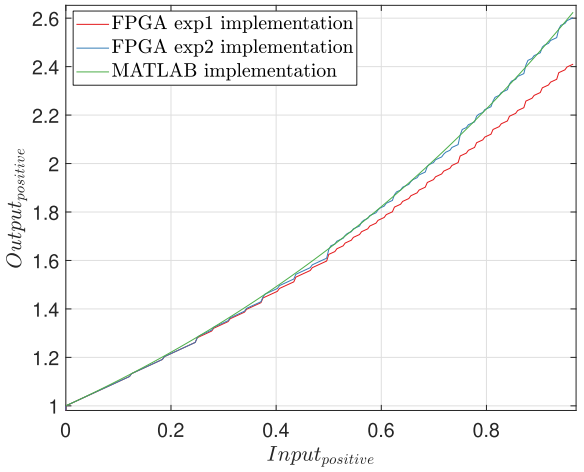


FIGURE 15. FPGA results exponential units 1 and 2 for positive input ranges.

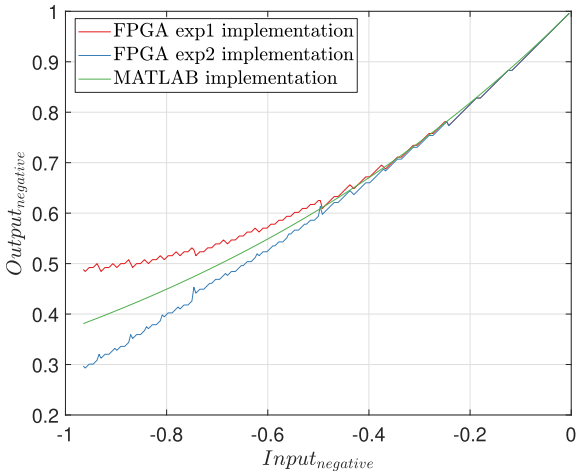


FIGURE 16. FPGA results exponential units 1 and 2 for negative input ranges.

TABLE 4. Comparative table showing the FPGA resources of different Exponential unit implementations.

Ref	bits	Registers	LUTs	DPS	Delay(ns)	FPGA
This design 1	17	0	186	0	1.9	Zynq 7Z007S
This design 2	17	0	541	0	2.1	Zynq 7Z007S
[44]	16	205	1301	0	-	Virtex-5
[43]	16	0	74	1	8	Virtex-7

In fact the high performance exponential unit has a delay 4 times less than in [43].

D. SD UNIT

Fig. 17, shows the results from the FPGA implementation of the design, here we can note that although not perfect the PWL *FPGA* implementation provides a sufficiently accurate estimation this can be seen in the zoomed window where the PWL tries to follow the unconstrained *MATLAB* implementation. In fact the MSE between the *MATLAB*, implementation

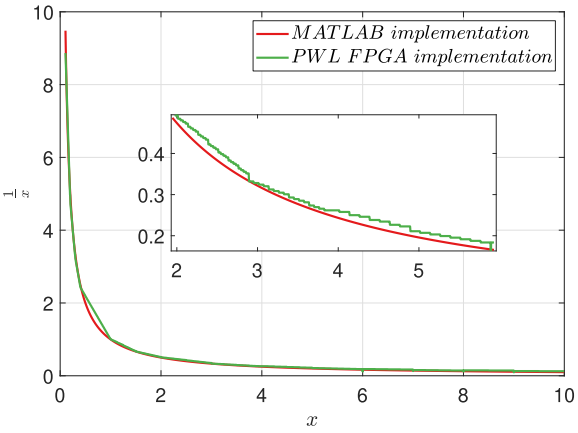


FIGURE 17. FPGA results SD unit.

TABLE 5. Comparative table showing the FPGA resources of different system division implementations.

Ref	bits	Registers	LUTs	DPS	Delay (ns)	FPGA
This design	17	0	290	0	5.2	Zynq 7Z007S
[45]	16	161	5592	0	4.2	Virtex-6

and reduced FPGA hardware is just 0.002. Table 1 (SD unit 1) shows the resources for for this first SD unit which utilises just over 500 LUTs for an input range of $x = [-10, 10]$, for the range $[-1, 1]$ the SD unit uses just 290 LUTs and hence the range should be predefined by the designer.

In contrast, from Table 1, SD unit 2 utilises just 635 mixed sized LUTs and produces an MSE of 3.212^{-04} for the range $-2 < x < 2$.

Table 5, shows a comparison between this implementation and another design with similar characteristics. Note that the number of resources are greatly reduced in this case whilst the critical delay remains almost the same.

E. SOFTMAX UNIT

Fig. 18, shows an example of the output from the *FPGA* compared to that of A unrestricted CPU *MATLAB* implementation for the varying input range of both inputs. Note that at the very start when both the input values are equal the prediction value is 0.5 as expected. As the distance between the inputs increases the prediction for input1 starts heading towards 1, reaching a maximum value of 0.61 when input2 = 0.49 and input 1 = 0.95.

With respect to the noise, Fig. 19, shows the absolute error between the two such that $|MATLAB - FPGA|$ is calculated, in this case we can note that the maximum absolute error is 0.011 and the mean absolute error is approximately 0.004. This error for most application specific applications should be sufficient however, as explained, increasing the bus widths and increasing fractional bus width will help in smoothing this function. The hardware requirements for the softmax function can be seen in Table 1 (softmax).

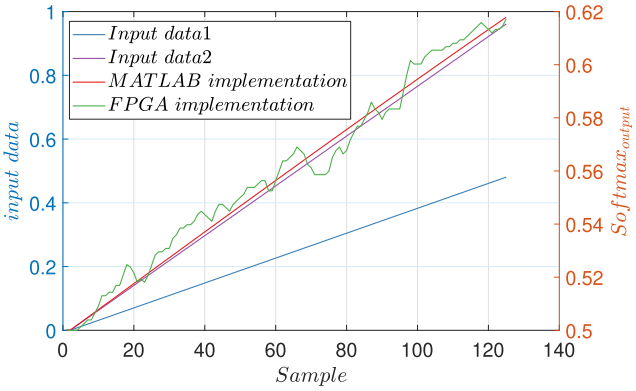


FIGURE 18. FPGA results softmax unit.

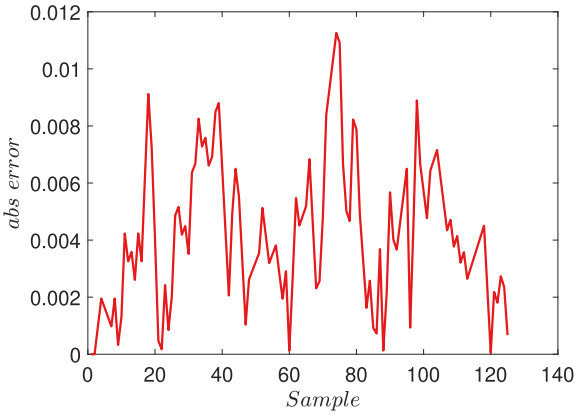


FIGURE 19. MSE MATLAB vs FPGA.

TABLE 6. Comparative table showing the FPGA resources of different softmax implementations.

Ref	bits	Registers	LUTs	DPS	Delay (ns)	FPGA
This design	17	0	573	0	4.6	Zynq 7Z007S
[36]	16	558	300	5	11	Virtex 6
[39]	16	16400	17870	0	8	Zynq 7
[46]	8(P=0)	1800	1580	0	3	Zynq ZC706

Furthermore, when compared to other softmax implementations as seen in table 6, our design uses less resources. This is mainly contributed to the effectiveness and low area consuming exponential function and SD unit implementations.

F. NN LAYER AND FEED BACK

The full feed forward resources for the NN shown in Fig. 2 can be seen in Table 1. The topology is 2,2,2,2 where the output layer is made up of softmax and is used to validate the block connections and provides a simple functionality verification by asking the NN to solve a simple validation input problem. The topology was chosen as a simple example of functionality. The full layer uses just over 3800 mixed-sized LUTs and a single perceptron as shown in Fig. 1, is implemented using just 197 mixed sized LUTs. The back

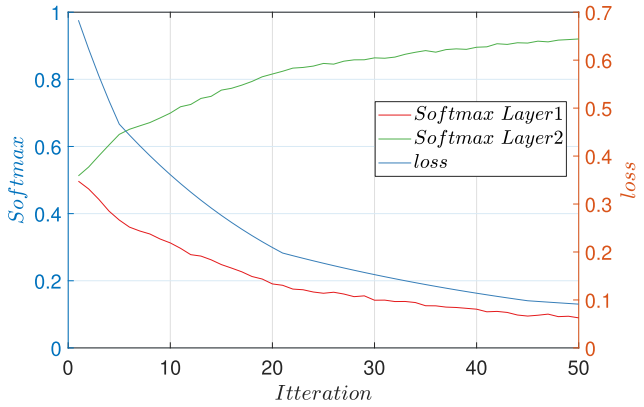


FIGURE 20. Softmax results test net.

pass including weight and bias actualisation via L1 normalisation uses 842.

To validate the network a simple test net setup was made with the input set as $Input = [-0.95, 0.95]$ and $\hat{y} = [0, 1]$. In this case the smoothed softmax outputs from the two layer neurons can be seen in Fig. 20, where layer 2 approaches 1 and layer 1 approaches 0. The loss function was calculated in *MATLAB* as the cross entropy loss and can be seen to rapidly converge to below 0.1 after 50 iterations.

The final MLP design has a critical path of approximately 3.12ns and maximum frequency of approximately 320MHz.

VI. ASIC IMPLEMENTATION

For comparison with other ASIC designs the individual blocks and full feed forward network from Section II-B were implemented in a 130 nm technology working at an operating voltage of 1.8 V. The technology was based on the *Skywater 130A* technology developed by Google [47], which consists of 5 V I/O pads and 5 metal layers, where the *ASIC* was implemented using the *VHDL* code in the software *OpenLane* [48].

Fig. 21, shows the power estimations of various blocks. The power estimations include leakage power, switching power and internal power. The frequency is based on the frequency of the input patterns generated to stimulate the circuits ranging from 10 KHz through to 100 MHz. From almost all of the designs we can note that the leakage power is almost negligible and is most notable in the total network of. This is consistent as the number of transistors increase. Taking a look at the maximum and minimum input frequencies we can note that at 100 MHz the RelU unit consumes negligible total power at approximately 0.0001 W which is to be expected due to small hardware requirements, at the same frequency the the first exponential unit consumes approximately 0.01 W which is around the same as the SM unit. The second exponential unit consumes a surprisingly high amount of power close to the total network of 0.1 W. The SD unit and softmax consume approximately 0.05 W each at 100 MHz. It is also worth noting that these power consumption's fall drastically with input frequency and at 10 KHz the entire FF network

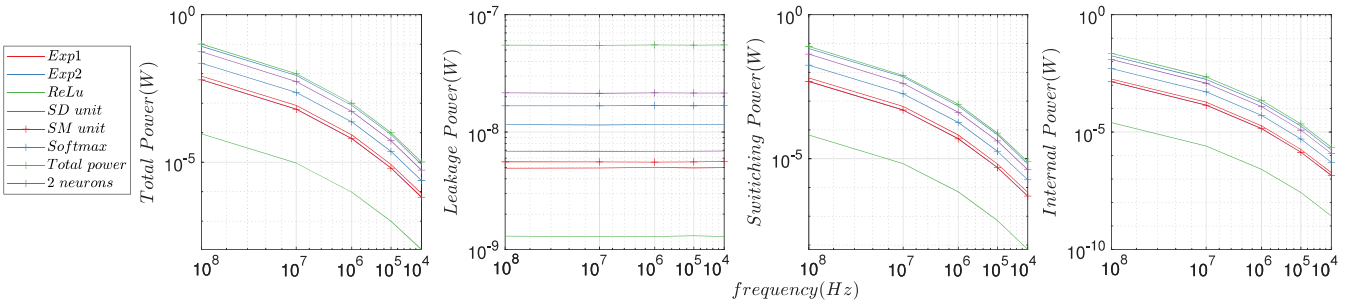


FIGURE 21. ASIC implementation power extraction results for designed blocks.

TABLE 7. Table showing a comparison between different ASIC implemented function estimation systems.

Processor	Algorithm	Active bits	Power(mW)	Area(mm^2)	Area(mm^2)[a]	f_o (MHz)	supply voltage	technology (nm)
			Multipliers					
This design	2^N shifts	17	0.07	0.016689	0.0083	1	1V8	130
[49]	R4ERBM	16	0.8	3.2	4.608	1	1V25	45
[50]	Booth	8	0.435	-	-	500	1V2	90
[51]	DLSB	16	6.432	6.863	4.95	100	1V2	90
			Exponential units					
This design unit1	TE	17	9	0.0181	0.009	100	1V8	130
[52]	TE analogue	8	-	0.0126	0.0045	0.1	0.65	180
[53]	CORDIC	16	0.602	0.18234	0.0658	10	1V	180
[54]	CORDIC	12	21.9	0.088	0.088	137	0.8V	65
			SD units					
This design	PWL	17 2's	0.8	0.013754	0.006877	1	1V8	130
[36]	CORDIC	16	0.51968	31.62	22.8	1	1V08	90
			Softmax					
This design	LUTs 2^N shifts	17 2's	0	0.024009	0.012	100	1V8	130
[37]	LUTs	16 (INT 8)	1.28	0.068439	0.049	100	1V	90
[46]	LUTs 2^N shifts	8	-	1	2.6	290	1V	28

[a] metric has been adjusted per technology and should be considered as $\frac{65\ nm}{technology} \cdot area$. Power has not been scaled to supply voltage
- indicates unknown or unspecified

consumes just 0.00001 W which is less than the ReLu unit alone at 1 MHz.

The total area breakdown of the implemented ASIC and individual components can be seen in the Fig. 22. Here we can note that the size of a single perceptron, from Section IV-C ooccupies a total area of approximately 0.036473 mm^2, which when scaled arbitrarily to a 65nm technology: $\frac{65\ nm}{130\ nm} = 0,0182365\ mm^2$. The total network as seen in Fig. 2, which includes 4 $\times$ perceptron units and 2$\times$ Softmax outputs and the feed back training network produce a total area of 0.121 mm^2 or 0.0605 mm^2, when scaled to the 65 nm technology and consumes 100 mW at 100 MHz input stimulus and consumes 0.01 mW at 10 KHz input stimulus.

Table 7, shows a more in-depth comparison to other similar implementations of the main components discussed here. From this we can deduce that the ASIC implementation of the multiplier uses less power per bits when compared to the state of the art. Nonetheless operating frequency and supply voltages are important factors to take into account. In such a case our multiplier performs more than 10 times better than in [49] when running at the same operating frequency. Furthermore, the area consumption of the design is well below that of the other designs despite the higher number of active bits. To re iterate this massive area performance is due to the hardwired arithmetic shifts. Carrying on we can see a similar scenario in exponential units, SD units, soft-max implementations and

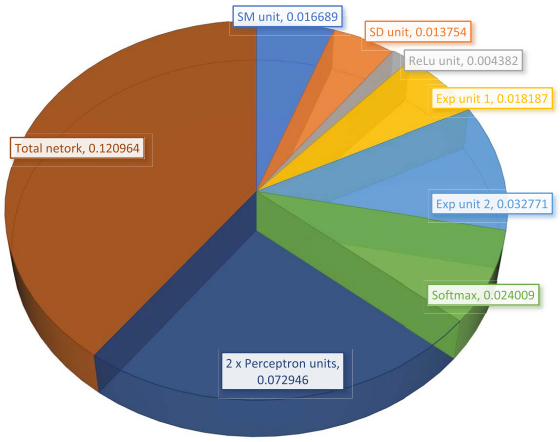


FIGURE 22. Area estimations based on technology in mm^2.

the overall size of a single perceptron. Note that the Relu and GD unit were not included as no solo ASIC implementations were found in the literature.

VII. CONCLUSION

In this manuscript the authors have designed power and area alleviating building blocks for ML applications without sacrificing throughput of the system. The building blocks were individually tested in a *FPGA* platform and compared to unrestricted *CPU* implementations of which compared very well producing small MSEs and reducing overall *FPGA* resources when compared to the state of the art. The building blocks were also implemented into a 130 nm *ASIC*, showing that the average power consumption for a two layer perceptron with ReLu activation and softmax output with feedback network was just 100 mW at 10^8 Hz input stimulus frequency and occupied an area of just 0.065 mm^2 when normalised to a 65 nm technology. Furthermore, the design leveraged no DSP blocks or BRAM modules reducing latencies and saving area via no generic hardware implementation.

APPENDIX
PD EQUATIONS

Starting at the last layer

$$\frac{\partial E}{\partial w_{1,2}} = \frac{\partial E}{\partial y_1} \cdot \frac{\partial y_1}{\partial N_{1,2,in}} \cdot \frac{\partial N_{1,2,in}}{\partial w_{1,2}} \tag{28}$$

$w_{3,2}$, can be calculated in a similar manner.

$$\frac{\partial E}{\partial w_{2,2}} = \frac{\partial E}{\partial y_2} \cdot \frac{\partial y_2}{\partial N_{2,2,in}} \cdot \frac{\partial N_{2,2,in}}{\partial w_{2,2}} \tag{29}$$

$w_{4,2}$, can be calculated in a similar manner.

Layer $h1$:

$$\frac{\partial E1}{\partial w_{1,1}} = \frac{\partial E1}{\partial y_1} \cdot \frac{\partial y_1}{\partial N_{1,2,in}} \cdot \frac{\partial N_{1,2,in}}{\partial N_{1,1,out}} \cdot \frac{\partial N_{1,1,out}}{\partial N_{1,1,in}} \cdot \frac{\partial N_{1,1,in}}{\partial w_{1,1}} \tag{30}$$

$$\frac{\partial E2}{\partial w_{1,1}} = \frac{\partial E2}{\partial y_2} \cdot \frac{\partial y_2}{\partial N_{2,2,in}} \cdot \frac{\partial N_{2,2,in}}{\partial N_{1,1,out}} \cdot \frac{\partial N_{1,1,out}}{\partial N_{1,1,in}} \cdot \frac{\partial N_{1,1,in}}{\partial w_{1,1}} \tag{31}$$

$$\frac{\partial E_{total}}{\partial w_{1,1}} = \frac{\partial E2}{\partial w_{1,1}} + \frac{\partial E1}{\partial w_{1,1}} \tag{32}$$

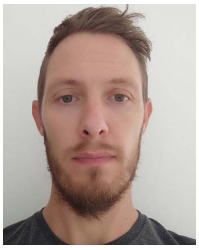
$w_{3,1}$, can be calculated in a similar manner.

Following the same procedure of back propagation $w_{2,1}$ and $w_{4,1}$ can be calculated.

REFERENCES

[1] A. Gilliam, *Machine Learning for Beginners: A History, a Basic Outline, and the Moral Quandary it Presents to Humankind*. CreateSpace Independent Publishing Platform, 2018, pp. 1–70.
[2] A. Burkov, *The_Hundred_Page_Machine_Learning_Book*, 2019, pp. 1–141.
[3] A. F. Hussein, N. Arunkumar, C. Gomes, A. K. Alzubaidi, Q. A. Habash, L. Santamaria-Granados, J. F. Mendoza-Moreno, and G. Ramirez-Gonzalez, "Focal and non-focal epilepsy localization: A review," *IEEE Access*, vol. 6, pp. 49306–49324, 2018.
[4] T. N. Alotaiby, S. A. Alshebeili, T. Alshawi, I. Ahmad, and F. E. A. El-Samie, "EEG seizure detection and prediction algorithms: A survey," *EURASIP J. Adv. Signal Process.*, vol. 2014, pp. 1–21, Dec. 2014, doi: 10.1186/1687-6180-2014-183.
[5] J. Yuan, X. Ran, K. Liu, C. Yao, Y. Yao, H. Wu, and Q. Liu, "Machine learning applications on neuroimaging for diagnosis and prognosis of epilepsy: A review," *J. Neurosci. Methods*, vol. 368, pp. 1–36, 2022.
[6] M. S. Kirschbaum, "Needs of parents of critically ill children," *Dimensions Crit. Care Nursing*, vol. 9, pp. 344–353, Nov. 1990.
[7] B. Abbasi and D. M. Goldenholz, "Machine learning applications in epilepsy," *Epilepsia*, vol. 60, no. 10, pp. 2037–2047, Oct. 2019.
[8] Y. Si, "Machine learning applications for electroencephalograph signals in epilepsy: A quick review," *Acta Epileptolog.*, vol. 2, no. 1, pp. 5–11, Dec. 2020.
[9] L. F. Nicolas-Alonso and J. Gomez-Gil, "Brain computer interfaces, a review," *Sensors*, vol. 12, no. 2, pp. 1211–1279, 2012.
[10] B. Ghafari, "Practical limits and challenges for powering of wireless systems in implantable and lab on a chip biomedical devices and review of the low power design techniques," in *Proc. IEEE Int. IoT, Electron. Mechatronics Conf. (IEMTRONICS)*, Apr. 2021, pp. 1–9.
[11] C. Serrano-Amenos, F. Hu, P. T. Wang, S. Kellis, R. A. Andersen, C. Y. Liu, P. Heydari, A. H. Do, and Z. Nenadic, "Thermal analysis of a skull implant in brain-computer interfaces," in *Proc. 42nd Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, Jul. 2020, pp. 3066–3069.
[12] G. O'Leary, D. M. Groppe, T. A. Valiante, N. Verma, and R. Genov, "NURIP: Neural interface processor for brain-state classification and programmable-waveform neurostimulation," *IEEE J. Solid-State Circuits*, vol. 53, no. 11, pp. 3150–3162, Nov. 2018.
[13] M. Delgado-Restituto, J. B. Romaine, and A. Rodriguez-Vazquez, "Phase synchronization operator for on-chip brain functional connectivity computation," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 5, pp. 957–970, Oct. 2019.
[14] K. Abdelhalim, V. Smolyakov, and R. Genov, "Phase-synchronization early epileptic seizure detector VLSI architecture," *IEEE Trans. Biomed. Circuits Syst.*, vol. 5, no. 5, pp. 430–438, Oct. 2011. [Online]. Available: http://ieeexplore.ieee.org/document/6046232/
[15] S. Venkatachalam and S.-B. Ko, "Design of power and area efficient approximate multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 5, pp. 1782–1786, May 2017.
[16] *Dsp48e1 Slice User Guide*, document UG479, Xilinx, San Jose, CA, USA, 2018, pp. 1–58. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
[17] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in *Proc. Chin. Control Decis. Conf. (CCDC)*, Jun. 2018, pp. 1836–1841, doi: 10.1109/CCDC.2018.8407425.
[18] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018, *arXiv:1811.03378*.
[19] S. Jadon, "A survey of loss functions for semantic segmentation," in *Proc. IEEE Conf. Comput. Intell. Bioinf. Comput. Biol. (CIBCB)*, Oct. 2020, pp. 1–7, doi: 10.1109/CIBCB48159.2020.9277638.
[20] Y. Deng, X. Zhou, J. Shen, G. Xiao, H. Hong, H. Lin, F. Wu, and B.-Q. Liao, "New methods based on back propagation (BP) and radial basis function (RBF) artificial neural networks (ANNs) for predicting the occurrence of haloketones in tap water," *Sci. Total Environ.*, vol. 772, Jun. 2021, Art. no. 145534.
[21] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.
[22] A. Sanaullah, C. Yang, Y. Alexeev, K. Yoshii, and M. C. Herbordt, "Real-time data analysis for medical diagnosis using FPGA-accelerated neural networks," *BMC Bioinf.*, vol. 19, no. S18, pp. 19–31, Dec. 2018, doi: 10.1186/s12859-018-2505-7.
[23] N. B. Gaikwad, V. Tiwari, A. Keskar, and N. C. Shivaprakash, "Efficient FPGA implementation of multilayer perceptron for real-time human activity classification," *IEEE Access*, vol. 7, pp. 26696–26706, 2019.
[24] J. Faraone, M. Kumm, M. Hardieck, P. Zipf, X. Liu, D. Boland, and P. H. W. Leong, "AddNet: Deep neural networks using FPGA-optimized multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 115–128, Jan. 2020.
[25] L. D. Medus, T. Iakymchuk, J. V. Frances-Villora, M. Bataller-Mompean, and A. Rosado-Munoz, "A novel systolic parallel hardware architecture for the FPGA acceleration of feedforward neural networks," *IEEE Access*, vol. 7, pp. 76084–76103, 2019.

[26] M. Bahoura, "FPGA implementation of blue whale calls classifier using high-level programming tool," *Electronics*, vol. 5, no. 1, p. 8, Feb. 2016.
[27] A. Suzuki, T. Morie, and H. Tamukoh, "A shared synapse architecture for efficient FPGA implementation of autoencoders," *PLoS ONE*, vol. 13, no. 3, Mar. 2018, Art. no. e0194049.
[28] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 317–328, Oct. 2020.
[29] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010.
[30] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *Proc. IEEE Int. Conf. Electron Devices Solid-State Circuits (EDSSC)*, Dec. 2010, pp. 1–4.
[31] Y. Ishiguchi, D. Isogai, T. Osawa, and S. Nakatake, "Analog perceptron circuit with DAC-based multiplier," *Integration*, vol. 63, pp. 240–247, Sep. 2018, doi: 10.1016/j.vlsi.2018.05.010.
[32] M. Bavandpour, M. R. Mahmoodi, and D. B. Strukov, "Energy-efficient time-domain vector-by-matrix multiplier for neurocomputing and beyond," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 9, pp. 1512–1516, Sep. 2019.
[33] T. Yang, Y. Wei, Z. Tu, H. Zeng, M. A. Kinsy, N. Zheng, and P. Ren, "Design space exploration of neural network activation function circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 10, pp. 1974–1978, Oct. 2019.
[34] W. Fu, J. Xia, X. Lin, M. Liu, and M. Wang, "Low-latency hardware implementation of high-precision hyperbolic functions Sinhx and Coshx based on improved CORDIC algorithm," *Electronics*, vol. 10, no. 20, p. 2533, Oct. 2021.
[35] J.-Y. Jhang, K.-H. Tang, C.-K. Huang, C.-J. Lin, and K.-Y. Young, "FPGA implementation of a functional neuro-fuzzy network for nonlinear system control," *Electronics*, vol. 7, no. 8, p. 145, Aug. 2018.
[36] M. K. Chughtai, M. B. Babar, M. Y. Qadri, and U. Qayyum, "A high speed and resource efficient approximation of softmax loss function," in *Proc. 16th Int. Bhurban Conf. Appl. Sci. Technol. (IBCAST)*, Jan. 2019, pp. 526–530.
[37] G. C. Cardarilli, L. D. Nunzio, R. Fazzolari, D. Giardino, A. Nannarelli, M. Re, and S. Spanò, "A pseudo-softmax function for hardware-based high speed image classification," *Sci. Rep.*, vol. 11, no. 1, pp. 1–10, Dec. 2021, doi: 10.1038/s41598-021-94691-7.
[38] H. M. H. Al-Rikabi, M. A. M. Al-Ja'afari, A. H. Ali, and S. H. Abdulwahed, "Generic model implementation of deep neural network activation functions using GWO-optimized SCPWL model on FPGA," *Microprocessors Microsyst.*, vol. 77, Sep. 2020, Art. no. 103141, doi: 10.1016/j.micpro.2020.103141.
[39] Q. Sun, Z. Di, Z. Lv, F. Song, Q. Xiang, Q. Feng, Y. Fan, X. Yu, and W. Wang, "A high speed SoftMax VLSI architecture based on basic-split," in *Proc. 14th IEEE Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, Oct. 2018, pp. 1–3.
[40] S. Ullah, S. Rehman, M. Shafique, and A. Kumar, "High-performance accurate and approximate multipliers for FPGA-based hardware accelerators," 2022. [Online]. Available: https://cfaed.tu-dresden.de/pd-downloads
[41] S. Venkatachalam and S.-B. Ko, "Design of power and area efficient approximate multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 5, pp. 1782–1786, May 2017.
[42] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, G. D. Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu, K. Pedro, N. Tran, E. Kreinar, S. Sagear, Z. Wu, and D. Hoang, "Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml," *Mach. Learning, Sci. Technol.*, vol. 2, no. 1, Dec. 2020, Art. no. 015001.
[43] J. Kim, V. Kornijcuk, and D. S. Jeong, "TS-EFA: Resource-efficient high-precision approximation of exponential functions based on template-scaling method," in *Proc. 21st Int. Symp. Qual. Electron. Design (ISQED)*, 2020, pp. 358–363, doi: 10.1109/ISQED48828.2020.9137012.
[44] S. Aggarwal, P. K. Meher, and K. Khare, "Concept, design, and implementation of reconfigurable CORDIC," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 4, pp. 1588–1592, 2016, doi: 10.1109/TVLSI.2015.2445855.
[45] S. Mopuri, S. Bhardwaj, and A. Acharyya, "Coordinate rotation-based design methodology for square root and division computation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 7, pp. 1227–1231, Jul. 2019.
[46] D. Zhu, S. Lu, M. Wang, J. Lin, and Z. Wang, "Efficient precision-adjustable architecture for softmax function in deep learning," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 12, pp. 3382–3386, Dec. 2020.
[47] *Google Sywater PDK*. Accessed: Apr. 14, 2022. [Online]. Available: https://github.com/google/skywater-pdk
[48] Openlane. *Openlane Eda Toolset*. Accessed: Apr. 22, 2022. [Online]. Available: https://github.com/The-OpenROAD-Project/OpenLane
[49] W. Liu, T. Cao, P. Yin, Y. Zhu, C. Wang, E. E. Swartzlander, and F. Lombardi, "Design and analysis of approximate redundant binary multipliers," *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 804–819, Jun. 2019.
[50] H. Xue, R. Patel, N. V. V. K. Boppana, and S. Ren, "Low-power-delay-product radix-4 8 × 8 booth multiplier in CMOS," *Electron. Lett.*, vol. 54, no. 6, pp. 344–346, Mar. 2018.
[51] V. Leon, S. Xydis, D. Soudris, and K. Pekmestzi, "Energy-efficient VLSI implementation of multipliers with double LSB operands," *IET Circuits, Devices Syst.*, vol. 13, no. 6, pp. 816–821, Sep. 2019.
[52] P. Srivastava and R. K. Sharma, "A novel pseudo-Taylor-exponential approximation technique for input–output range extension with reduced linearity error and its current-mode CMOS implementation," *Arabian J. Sci. Eng.*, vol. 46, no. 10, pp. 9809–9830, Oct. 2021, doi: 10.1007/s13369-021-05495-w.
[53] D. Wang, *Nanotoxicology Caenorhabditis Elegans*. Singapore: Springer, 2018, doi: 10.1007/978-981-13-0233-6.
[54] D. Li and D. Zhao, "High-throughput low-power area-efficient outphasing modulator based on unrolled and pipelined Radix-2 CORDIC," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 480–491, Feb. 2020.



JAMES BRIAN ROMAINE received the bachelor's degree in electronic engineering from the University of York, U.K., in 2013, the master's degree in microelectronic engineering from the University of Seville, Spain, in 2015, and the dual Ph.D. degree in physics from the University of Seville and the Consejo Superior de Investigaciones Científicas (CSIC). During his Ph.D. degree, he specialized in the implementation of ultra low-powered and area efficient application specific integrated circuits for the use in implantable medical devices. This had a special emphasis on epilepsy detection. He has published various works in respected journals, including IEEE TRANSACTIONS. He was awarded a prestigious FPI Grant from the state for his Ph.D. degree.

MARIO PEREIRA MARTÍN was born in Andalucia, Spain, in 1978. He received the degree in industrial engineering and the Ph.D. degree in automation, robotic and telecommunication from the University of Seville, Spain, in July 2009 and 2016, respectively. He joined the Predictive Group, Automation and System Engineering Department, University of Seville, in September 2009, where he has worked in several national and international research projects, combining his experience in the industrial electronic field with advanced control techniques. He joined the GEPOC Group, Automation and System Engineering Department, University of Seville, in 2013, where he helps in the development of new advanced control techniques. Since 2018, he has been an Assistant Professor with the Engineering Department, University of Loyola. His research interests include model predictive control, nonlinear control, multiparametric optimization, and networked systems. His research work has resulted in several articles in leading scientific journals and conferences.